

FILE1D**DBGEXT

E 15

DDDDDDDD DDDDDDDDD BBBBBBBBBB BBBBBBBBBB GGGGGGGG GGGGGGGG EEEEEEEEEE EEEEEEEEEE XX XX XX TTTTTTTT TTTTTT
DD DD BB BB GG EE XX XX XX TT
DD DD BB BB GG EE XX XX XX TT
DD DD BB BB GG EE XX XX XX TT
DD DD BB BB GG EE XX XX XX TT
DD DD BBBBBBBBBB GG EEEEEEEE XX XX TT
DD DD BBBBBBBBBB GG EEEEEEEE XX XX TT
DD DD BB BB GG GGGGGG EE XX XX XX TT
DD DD BB BB GG GGGGGG EE XX XX XX TT
DD DD BB BB GG GG EE XX XX XX TT
DD DD BB BB GG GG EE XX XX XX TT
DDDDDDDD DDDDDDDDD BBBBBBBBBB GGGGGG EEEEEEEEEE XX XX XX TT
DDDDDDDD DDDDDDDDD BBBBBBBBBB GGGGGG EEEEEEEEEE XX XX XX TT

The image shows a 10x10 grid of binary symbols. The symbols are arranged to form a stylized arrow pointing to the right. The 'L' symbols are located in the top-left corner, bottom-left corner, and a vertical column on the far left. The 'S' symbols are located in the top-right corner, bottom-right corner, and a diagonal line extending from the top-right towards the center. The remaining positions in the grid are empty.

F 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
\$255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 1
(1)

0001 0 ! DBGEXT.REQ
0002 0
0003 0 Version: 'V04-000'
0004 0
0005 0 *****
0006 0 *
0007 0 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0008 0 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0009 0 * ALL RIGHTS RESERVED.
0010 0 *
0011 0 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0012 0 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0013 0 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0014 0 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0015 0 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0016 0 * TRANSFERRED.
0017 0 *
0018 0 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0019 0 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0020 0 * CORPORATION.
0021 0 *
0022 0 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0023 0 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0024 0 *
0025 0 *
0026 0 *****
0027 0
0028 0 WRITTEN BY
0029 0 Rich Title October 1983
0030 0
0031 0 MODIFIED BY
0032 0
0033 0 Robert Conti November 2, 1983
0034 0 Edward Freedman December 12, 1983
0035 0
0036 0 MODULE FUNCTION
0037 0
0038 0 This module contains the definitions for the control blocks
0039 0 that are used in communications between DEBUG and the
0040 0 ADA multi-tasking run-time system. These same definitions will be
0041 0 extended for use in communication with the PPA multi-tasking
0042 0 system and other run-time systems, at a future time.
0043 0
0044 0
0045 0

0046 0

EXTERNAL CONTROL BLOCK

0047 0

An "External Control Block" is a data structure that can be used when DEBUG needs to call a routine that is not linked in as part of the DEBUG image.

0048 0

For example, DEBUG will have commands to support ADA multi-tasking. However, DEBUG has no knowledge of the workings of the ADA multi-tasking system and the data structures that describe tasks. Instead, DEBUG will call a routine in the ADA multitasking system in the course of processing SHOW TASK, SET TASK, or any other command that requires knowledge about tasks.

0049 0

There will be a single entry point, ADASDBGEXT, in the ADA multitasking system which is called by DEBUG. The External Control Block is the only parameter. Similarly, other multitasking run-time systems will have a single entry point, of the form <facility>\$DBGEXT, with the entry point taking an External Control Block as its single parameter. In general, the External Control Block can be used as a means of communication with run-time systems that are not part of DEBUG. For example, in debugging the language SCAN we may want to allow the user to set breakpoints on events such as a SCAN pattern-match. The External Control Block will be the data structure that we use to communicate with the SCAN run-time system.

0050 0

The DBGEXT\$V_FACILITY_ID field identifies which run-time system is being called. The VAX/VMS Facility code is used. Thus, it is assumed that there will be at most one DBGEXT entry point in the run-time code of any facility. Currently, legal values are the facility codes for ADA, PPA, and SCAN. This field may not actually be looked at (if desired, the run-time system may do a sanity check for the right value).

0051 0

Since there are several functions we want each run-time system to perform for us, there is a DBGEXT\$W_FUNCTION_CODE field which specifies which function is to be performed.

0052 0

All functions return a status code in the DBGEXT\$L_STATUS field. For all functions, there is a DBGEXT\$L_FLAGS field which can be used as a bitvector of flags. The exact use of these flags depends on the function.

0053 0

The use of the remaining fields of the data structure depends upon the "FACILITY_ID" field and upon the "FUNCTION_CODE" field.

0054 0

NOTE: DEBUG makes these calls with ASTs disabled. It is required that the run-time code not reenable ASTs during its execution.

0055 0

0056 0

0057 0

0058 0

0059 0

0060 0

0061 0

0062 0

0063 0

0064 0

0065 0

0066 0

0067 0

0068 0

0069 0

0070 0

0071 0

0072 0

0073 0

0074 0

0075 0

0076 0

0077 0

0078 0

0079 0

0080 0

0081 0

0082 0

0083 0

0084 0

0085 0

0086 0

0087 0

0088 0

0089 0

0090 0

0091 0

H 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
-\$2555DUA28:[DEBUG.SRC]DBGEXT.REQ;1 Page 3
(3)

0092 0 |
0093 0 | The following illustrates the header of an External Control Block.
0094 0 | The fields of an External Control Block are then illustrated
0095 0 | for the case where the "FACILITY_ID" is "ADA".
0096 0 |
0097 0 |
0098 0 | The following header is common to all External Control Blocks:
0099 0 |
0100 0 |-----+
0101 0 | unused : V_FACILITY_ID : DBGEXT\$W_FUNCTION_CODE |
0102 0 |-----+
0103 0 | 1 :-----+
0104 0 |-----+
0105 0 | 2 :-----+
0106 0 | (some flags unused) : DBGEXT\$L_FLAGS |
0107 0 |-----+
0108 0 | 3 :-----+
0109 0 | reserved for future use |
0110 0 |-----+

I 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 v4.0-742
\$255SDUA2B:[DEBUG.SRC]DBGEXT.REQ:1 Page 4 (4)

0110 0 | The following illustrates the control block when the FACILITY_ID
0111 0 | field is "ADA". This control block is used
0112 0 | for most functions (some functions, e.g. GET_REGISTERS and SET_REGISTERS
0113 0 | use a longer control block, displayed later).

```
+-----+  
0 |unused : V_FACILITY_ID : DBGEXT$W_FUNCTION_CODE|  
+-----+  
1 |DBGEXT$L_STATUS|  
+-----+  
2 |(some flags unused) DBGEXT$L_FLAGS|  
+-----+  
3 |reserved for future use|  
+-----+  
4 |DBGEXT$L_TASK_VALUE|  
+-----+  
5 |DBGEXT$L_TASK_NUMBER|  
+-----+  
6 |unused : V_HOLD: V_STATE : DBGEXT$W_SPECIFIED_FLAGS|  
+-----+  
7 |DBGEXT$V_PRIORITY|  
+-----+  
8 |DBGEXT$L_PRINT_ROUTINE|  
+-----+  
9 |DBGEXT$L_EVENT_ID|  
+-----+
```

J 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
\$255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 5
(5)

0137 0
0138 0 The following fields are present when the "FACILITY_ID" field is
0139 0 "ADA" and the function code is
0140 0 DBGEXTSK_GET_REGISTERS.
0141 0 DBGEXTSK_SET_REGISTERS.
0142 0 DBGEXTSK_SET_ACTIVE.
0143 0 For all other functions, the smaller block (without the register fields)
0144 0 is passed in.
0145 0

```
0 unused : V_FACILITY_ID : DBGEXTSW_FUNCTION_CODE
+-----+
1           DBGEXTSL_STATUS
+-----+
2 (some flags unused)   DBGEXTSL_FLAGS
+-----+
3             reserved for future use
+-----+
4           DBGEXTSL_TASK_VALUE
+-----+
5           DBGEXTSL_TASK_NUMBER
+-----+
6 unused :V_HOLD: V_STATE : DBGEXTSW_SPECIFIED_FLAGS
+-----+
7           DBGEXTSV_PRIORITY
+-----+
8           DBGEXTSL_PRINT_ROUTINE
+-----+
9           DBGEXTSL_EVENT_ID
+-----+
10          DBGEXTSL_R0
+-----+
11          DBGEXTSL_R1
+-----+
12          DBGEXTSL_R2
+-----+
13          DBGEXTSL_R3
+-----+
14          DBGEXTSL_R4
+-----+
15          DBGEXTSL_R5
+-----+
16          DBGEXTSL_R6
+-----+
17          DBGEXTSL_R7
+-----+
18          DBGEXTSL_R8
+-----+
19          DBGEXTSL_R9
+-----+
20          DBGEXTSL_R10
+-----+
21          DBGEXTSL_R11
+-----+
22          DBGEXTSL_AP
+-----+
23          DBGEXTSL_FP
```

K 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 v4.0-742
\$255SDUA28:[DEBUG.SRC]DBGEXT.REQ:1 Page 6 (5)

0194	0	24	----- DBGEXTSL_SP -----
0195	0	25	----- DBGEXTSL_PC -----
0196	0	26	----- DBGEXTSL_PSL -----
0197	0		
0198	0		
0199	0		
0200	0		
0201	0		
0202	0		

L 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ:1 Page 7
(6)

0203 0 |+
0204 0 |
0205 0 |
0206 0 |
0207 0 | FIELD DBGEXTSHEADER_FIELDS =
0208 0 | SET
0209 0 | DBGEXT\$W_FUNCTION_CODE = [0, 0, 16, 0],
0210 0 | DBGEXT\$V_FACILITY_ID = [0, 16, 12, 0],
0211 0 | ! reserved = [0, 28, 0],
0212 0 | DBGEXTSL_STATUS = [1, 0, 32, 0],
0213 0 |
0214 0 | DBGEXTSL_FLAGS = [2, 0, 32, 0],
0215 0 | DBGEXT\$V_ALL = [2, 0, 1, 0],%((WHAT WILL ALL DO?-tbs))%
0216 0 | DBGEXT\$V_FULL = [2, 1, 1, 0],%((explain FULL -tbs))%
0217 0 |
0218 0 | DBGEXT\$V_PSEUDO_GO = [2, 2, 1, 0],
0219 0 | ! Pseudo-go is set by the run-time system on return to DEBUG to
0220 0 | indicate that DEBUG must do a pseudo-GO to accomplish the function.
0221 0 | Used only for function SET_ACTIVE (see discussion under SET_ACTIVE).
0222 0 |
0223 0 | DBGEXT\$V_NO_HEADER = [2, 3, 1, 0]
0224 0 | ! Suppresses output of headers on a SHOW_TASK, SHOW_STATISTICS,
0225 0 | or SHOW_DEADLOCKS.
0226 0 |
0227 0 | ! reserved = [0, 4, 28, 0],
0228 0 | ! reserved = [4, 0, 32, 0],
0229 0 TES:
0230 0
0231 0
0232 0 | FIELD DBGEXTSADA_FIELDS =
0233 0 | SET
0234 0 | DBGEXTSL_TASK_VALUE = [4, 0, 32, 0],
0235 0 | DBGEXTSL_TASK_NUMBER = [5, 0, 32, 0],
0236 0 | DBGEXT\$W_SPECIFIED_FLAGS = [6, 0, 16, 0],
0237 0 | DBGEXT\$V_HOLD_SPECIFIED = [6, 0, 1, 0],
0238 0 | DBGEXT\$V_STATE_SPECIFIED = [6, 1, 1, 0],
0239 0 | DBGEXT\$V_PRIORITY_SPECIFIED = [6, 2, 1, 0],
0240 0 | ! reserved = [6, 3, 13, 0],
0241 0 | DBGEXT\$V_STATE = [6, 16, 4, 0],
0242 0 | DBGEXT\$V_STATE_RUNNING = [6, 16, 1, 0],
0243 0 | DBGEXT\$V_STATE_READY = [6, 17, 1, 0],
0244 0 | DBGEXT\$V_STATE_SUSPENDED = [6, 18, 1, 0],
0245 0 | DBGEXT\$V_STATE_TERMINATED = [6, 19, 1, 0],
0246 0 | DBGEXT\$V_HOLD = [6, 20, 1, 0],
0247 0 | ! reserved = [6, 21, 11, 0],
0248 0 | DBGEXTSL_PRIORITY = [7, 0, 32, 0],
0249 0 | DBGEXT\$V_PRIORITY_00 = [7, 0, 1, 0],
0250 0 | DBGEXT\$V_PRIORITY_01 = [7, 1, 1, 0],
0251 0 | DBGEXT\$V_PRIORITY_02 = [7, 2, 1, 0],
0252 0 | DBGEXT\$V_PRIORITY_03 = [7, 3, 1, 0],
0253 0 | DBGEXT\$V_PRIORITY_04 = [7, 4, 1, 0],
0254 0 | DBGEXT\$V_PRIORITY_05 = [7, 5, 1, 0],
0255 0 | DBGEXT\$V_PRIORITY_06 = [7, 6, 1, 0],
0256 0 | DBGEXT\$V_PRIORITY_07 = [7, 7, 1, 0],
0257 0 | DBGEXT\$V_PRIORITY_08 = [7, 8, 1, 0],
0258 0 | DBGEXT\$V_PRIORITY_09 = [7, 9, 1, 0],
0259 0 | DBGEXT\$V_PRIORITY_10 = [7, 10, 1, 0],

M 15

15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742

\$255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 8
(6)

```

0260 0      DBGEXTSV_PRIORITY_11      = [ 7, 11, 1, 0],
0261 0      DBGEXTSV_PRIORITY_12      = [ 7, 12, 1, 0],
0262 0      DBGEXTSV_PRIORITY_13      = [ 7, 13, 1, 0],
0263 0      DBGEXTSV_PRIORITY_14      = [ 7, 14, 1, 0],
0264 0      DBGEXTSV_PRIORITY_15      = [ 7, 15, 1, 0],
0265 0      DBGEXTSV_PRIORITY_16      = [ 7, 16, 1, 0],
0266 0      DBGEXTSV_PRIORITY_17      = [ 7, 17, 1, 0],
0267 0      DBGEXTSV_PRIORITY_18      = [ 7, 18, 1, 0],
0268 0      DBGEXTSV_PRIORITY_19      = [ 7, 19, 1, 0],
0269 0      DBGEXTSV_PRIORITY_20      = [ 7, 20, 1, 0],
0270 0      DBGEXTSV_PRIORITY_21      = [ 7, 21, 1, 0],
0271 0      DBGEXTSV_PRIORITY_22      = [ 7, 22, 1, 0],
0272 0      DBGEXTSV_PRIORITY_23      = [ 7, 23, 1, 0],
0273 0      DBGEXTSV_PRIORITY_24      = [ 7, 24, 1, 0],
0274 0      DBGEXTSV_PRIORITY_25      = [ 7, 25, 1, 0],
0275 0      DBGEXTSV_PRIORITY_26      = [ 7, 26, 1, 0],
0276 0      DBGEXTSV_PRIORITY_27      = [ 7, 27, 1, 0],
0277 0      DBGEXTSV_PRIORITY_28      = [ 7, 28, 1, 0],
0278 0      DBGEXTSV_PRIORITY_29      = [ 7, 29, 1, 0],
0279 0      DBGEXTSV_PRIORITY_30      = [ 7, 30, 1, 0],
0280 0      DBGEXTSV_PRIORITY_31      = [ 7, 31, 1, 0],
0281 0      DBGEXTSL_PRINT_ROUTINE   = [ 8, 0, 32, 0],
0282 0      DBGEXTSL_EVENT_ID       = [ 9, 0, 32, 0],
0283 0      TES;

0284 0
0285 0      FIELD DBGEXTSREG_FIELDS =
0286 0          SET
0287 0          DBGEXTSL_R0      = [10, 0, 32, 0],
0288 0          DBGEXTSL_R1      = [11, 0, 32, 0],
0289 0          DBGEXTSL_R2      = [12, 0, 32, 0],
0290 0          DBGEXTSL_R3      = [13, 0, 32, 0],
0291 0          DBGEXTSL_R4      = [14, 0, 32, 0],
0292 0          DBGEXTSL_R5      = [15, 0, 32, 0],
0293 0          DBGEXTSL_R6      = [16, 0, 32, 0],
0294 0          DBGEXTSL_R7      = [17, 0, 32, 0],
0295 0          DBGEXTSL_R8      = [18, 0, 32, 0],
0296 0          DBGEXTSL_R9      = [19, 0, 32, 0],
0297 0          DBGEXTSL_R10     = [20, 0, 32, 0],
0298 0          DBGEXTSL_R11     = [21, 0, 32, 0],
0299 0          DBGEXTSL_AP      = [22, 0, 32, 0],
0300 0          DBGEXTSL_FP      = [23, 0, 32, 0],
0301 0          DBGEXTSL_SP      = [24, 0, 32, 0],
0302 0          DBGEXTSL_PC      = [25, 0, 32, 0],
0303 0          DBGEXTSL_PSL     = [26, 0, 32, 0],
0304 0          TES;

0305 0
0306 0          LITERAL
0307 0          DBGEXTSK_HEADER_SIZE  = 4    ! Size of header in longwords
0308 0          DBGEXTSK_ADA_SIZE1   = 16   ! Size of block for ADA (without regs)
0309 0          DBGEXTSK_ADA_SIZE2   = 27   ! Size of block for ADA (with regs)
0310 0          DBGEXTSK_MAX_SIZE    = 27   ! Max of above sizes
0311 0
0312 0          MACRO
0313 0          DBGEXTSCONTROL_BLOCK = BLOCK [DBGEXTSK_MAX_SIZE]
0314 0          FIELD ( DBGEXTSHEADER_FIELDS,
0315 0                  DBGEXTSADA_FIELDS,
0316 0                  DBGEXTSREG_FIELDS ) %:

```

N 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
\$255\$DUA28:[DEBUG.SRC]DBGEXT.REQ:1 Page 9
(6)

: 0317 0

```

0318 0
0319 0
0320 0
0321 0
0322 0
0323 0
0324 0
0325 0
0326 0
0327 0
0328 0
0329 0
0330 0
0331 0
0332 0
0333 0
0334 0
0335 0
0336 0
0337 0
0338 0
0339 0
0340 0
0341 0
0342 0
0343 0
0344 0
0345 0
0346 0
0347 0
0348 0
0349 0
0350 0
0351 0
0352 0
0353 0
0354 0
0355 0
0356 0
0357 0
0358 0
0359 0
0360 0
0361 0
0362 0
0363 0
0364 0
0365 0
0366 0
0367 0
0368 0
0369 0
0370 0
0371 0
0372 0
0373 0
0374 0

;+
;+ Generally, multiple priorities and states are valid as input when calling
;+ the ADA run time system but are not valid as output values on return from
;+ the call. Therefore, the following constants are provided for convenience
;+ in setting and testing the contents of the fields DBGEXTSV STATE and
;+ DBGEXTSV_PRIORITY. They define the only possible values of the respective
;+ fields when multiple priorities and states are not allowed.
;+ Constants for DBGEXTSV_HOLD are provided for completeness.

LITERAL
DBGEXT$K_MIN_STATE = .
DBGEXT$K_MAX_STATE = . %((superfluous? -tbs))%

DBGEXT$S_STATE = 4. ! size of DBGEXTSV_STATE
DBGEXT$K_STATE_RUNNING = 1 ^ 0. ! values for DBGEXTSV_STATE
DBGEXT$K_STATE_READY = 1 ^ 1.
DBGEXT$K_STATE_SUSPENDED = 1 ^ 2.
DBGEXT$K_STATE_TERMINATED = 1 ^ 3.

DBGEXT$S_HOLD = 1. ! size of DBGEXTSV_HOLD
DBGEXT$K_HOLD = 1 ^ 0. ! values for DBGEXTSV_HOLD

DBGEXT$S_PRIORITY = 32. ! size of DBGEXTSV_PRIORITY
DBGEXT$K_PRIORITY_00 = 1 ^ 0. ! values for DBGEXTSV_PRIORITY
DBGEXT$K_PRIORITY_01 = 1 ^ 1.
DBGEXT$K_PRIORITY_02 = 1 ^ 2.
DBGEXT$K_PRIORITY_03 = 1 ^ 3.
DBGEXT$K_PRIORITY_04 = 1 ^ 4.
DBGEXT$K_PRIORITY_05 = 1 ^ 5.
DBGEXT$K_PRIORITY_06 = 1 ^ 6.
DBGEXT$K_PRIORITY_07 = 1 ^ 7.
DBGEXT$K_PRIORITY_08 = 1 ^ 8.
DBGEXT$K_PRIORITY_09 = 1 ^ 9.
DBGEXT$K_PRIORITY_10 = 1 ^ 10.
DBGEXT$K_PRIORITY_11 = 1 ^ 11.
DBGEXT$K_PRIORITY_12 = 1 ^ 12.
DBGEXT$K_PRIORITY_13 = 1 ^ 13.
DBGEXT$K_PRIORITY_14 = 1 ^ 14.
DBGEXT$K_PRIORITY_15 = 1 ^ 15.
DBGEXT$K_PRIORITY_16 = 1 ^ 16.
DBGEXT$K_PRIORITY_17 = 1 ^ 17.
DBGEXT$K_PRIORITY_18 = 1 ^ 18.
DBGEXT$K_PRIORITY_19 = 1 ^ 19.
DBGEXT$K_PRIORITY_20 = 1 ^ 20.
DBGEXT$K_PRIORITY_21 = 1 ^ 21.
DBGEXT$K_PRIORITY_22 = 1 ^ 22.
DBGEXT$K_PRIORITY_23 = 1 ^ 23.
DBGEXT$K_PRIORITY_24 = 1 ^ 24.
DBGEXT$K_PRIORITY_25 = 1 ^ 25.
DBGEXT$K_PRIORITY_26 = 1 ^ 26.
DBGEXT$K_PRIORITY_27 = 1 ^ 27.
DBGEXT$K_PRIORITY_28 = 1 ^ 28.
DBGEXT$K_PRIORITY_29 = 1 ^ 29.
DBGEXT$K_PRIORITY_30 = 1 ^ 30.
DBGEXT$K_PRIORITY_31 = 1 ^ 31.

```

{ 16
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_8255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 11
(8)

0375 0
0376 0
0377 0
0378 0
0379 0
0380 0
0381 0
0382 0
0383 0
0384 0
0385 0
0386 0
0387 0
0388 0
0389 0

FACILITY CODES

The following are the possible values of the DBGEXT\$V.FACILITY_ID field.
These correspond to the different run-time system we are
communicating with.

ADAS_FACILITY
PPAS_FACILITY
SCNS_FACILITY

: *QUEST%((-tbs))%
: Do PPA and SCAN have facility mnemonics and codes? Are the
: above guesses correct?

0 16
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 12 (9)

0390 0
0391 0
0392 0
0393 0
0394 0
0395 0
0396 0
0397 0
0398 0
0399 0
0400 0
0401 0
0402 0
0403 0
0404 0
0405 0
0406 0
0407 0
0408 0
0409 0
0410 0
0411 0
0412 0
0413 0
0414 0
0415 0
0416 0
0417 0
0418 0
0419 0
0420 0
0421 0
0422 0
0423 0
0424 0
0425 0
0426 0
0427 0
0428 0
0429 0
0430 0
0431 0
0432 0
0433 0
0434 0
0435 0
0436 0

FUNCTION CODES

The following are the possible values of the DBGEXTSW_FUNCTION_CODE field when the contents of the FACILITY_ID field is ADASFACILITY. These correspond to the functions that the ADA run-time system will be asked to perform.

Summary of the defined Function codes

DBGEXTSK_MIN_FUNCT = 1. ! For CASE bounds

These are used to obtain and convert task values

DBGEXTSK_CVT_VALUE_NUM = 1.
DBGEXTSK_CVT_NUM_VALUE = 2.
DBGEXTSK_NEXT_TASK = 3.

These are used to ask ADA to display task information

DBGEXTSK_SHOW_TASK = 4.
DBGEXTSK_SHOW_STATISTICS = 5.
DBGEXTSK_SHOW_DEADLOCK = 6.

These are used to get and set various attributes of one or more tasks

Task state
DBGEXTSK_GET_STATE = 7.
DBGEXTSK_SET_ACTIVE = 8.
DBGEXTSK_SET_TERMINATE = 9.
DBGEXTSK_SET_HOLD = 10.
DBGEXTSK_SET_MOLD = 11.

Task priority
DBGEXTSK_GET_PRIORITY = 12.
DBGEXTSK_SET_PRIORITY = 13.
DBGEXTSK_RESTORE_PRIORITY = 14.

Task registers
DBGEXTSK_GET_REGISTERS = 15.
DBGEXTSK_SET_REGISTERS = 16.

These are used to control definable events

DBGEXTSK_ENABLE_EVENT = 17.
DBGEXTSK_DISABLE_EVENT = 18.

DBGEXTSK_MAX_FUNCT = 18: ! For CASE bounds

0437 0

LITERAL

| A minimum task code is defined for CASE statement bounds.

DBGEXTSK_MIN_FUNCT = 1.

| CVT_VALUE_NUM takes a task value and converts it to a task number.

INPUT - The task value is placed in the DBGEXTSL_TASK_VALUE field.

OUTPUT - The task number is returned in the DBGEXTSL_TASK_NUMBER field.

(If the task does not exist, this function returns
status STSSK_SEVERE).%((TASK DOES NOT EXIST CODE? -tbs))%
%((VALUE IS NOT LEGAL OR ACCVIO? -tbs))%

DBGEXTSK_CVT_VALUE_NUM = 1.

| CVT_NUM_VALUE takes a task number and converts it to a task value.

INPUT - The task number is placed in the DBGEXTSL_TASK_NUMBER field.

OUTPUT - The task value is returned in the DBGEXTSL_TASK_VALUE field.

(If the task does not exist, this function returns
status STSSK_SEVERE).%((TASK DOES NOT EXIST CODE? -tbs))%

DBGEXTSK_CVT_NUM_VALUE = 2.

| NEXT_TASK gives a task value and asks ADA to specify the "next" task.
The ordering of tasks is up to the ADA run-time system. The only
requirement on order is that if we start with any task, and repeatedly
ask for the "next" without giving the user program control in between,
then we will cycle through all the tasks and return to the task we
started with. If selection criteria are imposed, then we will cycle
through all tasks which match that criteria.

INPUTS - The task value is placed in the DBGEXTSL_TASK_VALUE field.

If the TASK VALUE field is zero (implying the NULL task) the
next task will be the main task of the program.

The ALL flag is ignored, ADA will consider it on by default.

The set of tasks to cycle through can be restricted by
imposing a selection criteria. The PRIORITY, and/or STATE,
and/or HOLD fields can contain values which a task must match
to be part of the set (e.g. SHOW TASK/PRI=3/HOLD/STATE=READY).
When such a restriction is desired, the DBGEXTSV_xxx SPECIFIED
bits must be set accordingly. If no restriction is desired,
the SPECIFIED bits must be zero. A task must match all the
criteria which are specified to be part of the set.

%((Multiple PRI and STATE can be given as these are bit fields -tbs))%

0494 0
0495 0
0496 0
0497 0
0498 0
0499 0
0500 0
0501 0
0502 0
0503 0
0504 0
0505 0
0506 0
0507 0
0508 0
0509 0
0510 0
0511 0
0512 0
0513 0
0514 0
0515 0
0516 0
0517 0
0518 0
0519 0
0520 0
0521 0
0522 0
0523 0
0524 0
0525 0
0526 0
0527 0
0528 0
0529 0
0530 0
0531 0
0532 0
0533 0
0534 0
0535 0
0536 0
0537 0
0538 0
0539 0
0540 0
0541 0
0542 0
0543 0
0544 0
0545 0
0546 0
0547 0
0548 0
0549 0
0550 0

OUTPUT - The "next" task value is returned in `DBGEXTSL_TASK_VALUE`.
`DBGEXTSK_NEXT_TASK` = 3.

SHOW TASK is used to request that ADA display information about a specified task.

INPUTS - The task value is placed in the `DBGEXTSL_TASK_VALUE` field.
The address of a print routine that ADA is to call, to display the information, is placed in the field `DBGEXTSL_PRINT_ROUTINE` (see `DBG$PRINT_ROUTINE` below).
If the `DBGEXTSV_FULL` bit is set, more detailed information is displayed.

OUTPUT - none.

`DBGEXTSK_SHOW_TASK` = 4.

SHOW_STATISTICS requests that the ADA run-time system display statistics about the overall state of the multitasking system.

INPUTS - The address of a print routine is given in the field `DBGEXTSL_PRINT_ROUTINE`.
If the `DBGEXTSV_FULL` bit is set, more detailed information is displayed.

OUTPUT - none.

`DBGEXTSK_SHOW_STAT` = 5.

SHOW_DEADLOCK requests that the ADA run-time system display information about deadlocks within the multitasking system.

INPUTS - The address of a print routine is given in the field `DBGEXTSL_PRINT_ROUTINE`.
If the `DBGEXTSV_FULL` bit is set, more detailed information is displayed.

OUTPUT - none.

`DBGEXTSK_SHOW_DEADLOCK` = 6.

GET STATE inquires about the "state" and HOLD condition of a task. The "state" can be one of RUNNING, READY, SUSPENDED, TERMINATED. The state codes are defined below.

INPUT - The task value is placed in the `DBGEXTSL_TASK_VALUE` field.

0551 0
0552 0
0553 0
0554 0
0555 0
0556 0
0557 0
0558 0
0559 0
0560 0
0561 0
0562 0
0563 0
0564 0
0565 0
0566 0
0567 0
0568 0
0569 0
0570 0
0571 0
0572 0
0573 0
0574 0
0575 0
0576 0
0577 0
0578 0
0579 0
0580 0
0581 0
0582 0
0583 0
0584 0
0585 0
0586 0
0587 0
0588 0
0589 0
0590 0
0591 0
0592 0
0593 0
0594 0
0595 0
0596 0
0597 0
0598 0
0599 0
0600 0
0601 0
0602 0
0603 0
0604 0
0605 0
0606 0
0607 0

| OUTPUTS - A code representing the state is returned in the %((V_STATE -tbs))%
| DBGEXTSV_STATE field.
| The DBGEXTSV_HOLD field is also set if the task is on HOLD.
| DBGEXTSK_GET_STATE = 7.

| GET_ACTIVE obtains the task value of the active task.
| (The active task is that task in whose context (stack and register set)
| DEBUG is executing. This is contrasted with the "visible task" --
| the task whose register set is temporarily in use by DEBUG
| as a default for the purposes of SHOW CALLS, EXAMINE, etc.).
| INPUTS - none
| OUTPUT - The task value of the active task is returned
| in DBGEXTSL_TASK_VALUE.
| %((Can the active task be the null task? -tbs))%
| DBGEXTSK_GET_ACTIVE = 8.

| SET_ACTIVE requests the run-time system to switch the active
| task to that given in DBGEXTSL_TASK_VALUE. The "long form" DEBUG
| control block is used. The registers provided by DEBUG in the control
| block are those of the (currently) active task. The run-time
| system uses these to save the registers of the active task. It
| may also modify this register set, (currently only the PC and PSL).
| When this call returns, DEBUG should use the possibly-modified
| register values as the active register set. If the PSEUDO GO bit
| is set, DEBUG should then perform the actions of a normal GO,
| except that ASTs are left disabled. This "pseudo-GO"
| will enter special run-time code that will switch-out the
| currently active task, switch-in the requested active task, and
| reinvoke DEBUG in that task. (A special event code is assigned
| to this "reinvoke DEBUG event". The reinvocation event signifies
| to DEBUG that certain components of its state are to be
| gotten from values saved from DEBUG's prior incarnation, not those
| at the reinvocation event. One such saved state component is
| the "AST enablement" status - whether ASTs were enabled when
| DEBUG was invoked.)
| Despite these gyrations, to the user typing
| DEBUG> SET TASK/ACTIVE T1, it appears he has entered a simple command
| immediately followed by a DEBUG> prompt.
| INPUTS - The task value of the to-become-active task is set
| in DBGEXTSL_TASK_VALUE.
| The registers of the (currently) active task are stored in
| fields DBGEXTSL_R0 through DBGEXTSL_PSL.
| OUTPUTS - The register set of the new active task is modified
| by the run-time system, in DBGEXTSL_R0 through DBGEXTSL_PSL.

0608 0 |
0609 0 | The DBGEXTSV_PSEUDO_GO flag may be set, in which case,
0610 0 | DEBUG should perform a "pseudo go" operation.
0611 0 |
0612 0 | DBGEXTSK_SET_ACTIVE = 9.
0613 0 |
0614 0 | SET_TERMINATE is used to cause ADA to terminate a task. It is used
0615 0 | to implement the command SET TASK/TERMINATE.
0616 0 |
0617 0 | INPUTS - The task value is placed in the DBGEXTSL_TASK_VALUE field.
0618 0 |
0619 0 | If the TASK_VALUE field is zero and the ALL flag
0620 0 | is set, then the function is done for all tasks.
0621 0 |
0622 0 | OUTPUT - none
0623 0 |
0624 0 | DBGEXTSK_SET_TERMINATE = 10.
0625 0 |
0626 0 |
0627 0 | SET_HOLD is used to put a task on hold or to release a task that was
0628 0 | previously put on hold. It is used to implement the command
0629 0 | SET TASK/HOLD which leaves the state of a task as-is, except that each
0630 0 | task is marked HOLD.
0631 0 |
0632 0 | INPUTS - The task value is placed in the DBGEXTSL_TASK_VALUE field.
0633 0 |
0634 0 | If the TASK_VALUE field is zero and the ALL flag
0635 0 | is set, then the function is done for all tasks.
0636 0 |
0637 0 | %% Will the /ALL selection criteria be used for the SET_xxx codes? -tbs)%%
0638 0 |
0639 0 | The desired status of HOLD is placed into the
0640 0 | DBGEXTSV_HOLD field. (1 => HOLD, 0 => RELEASE)
0641 0 |
0642 0 | %% Is the request 1=>1 or 0=>0 legal? -tbs)%%
0643 0 |
0644 0 | OUTPUT - none
0645 0 |
0646 0 | DBGEXTSK_SET_HOLD = 11.
0647 0 |
0648 0 |
0649 0 | GET_PRIORITY inquires about the priority of a specified task.
0650 0 |
0651 0 | INPUT - The task value is placed in the DBGEXTSL_TASK_VALUE field.
0652 0 |
0653 0 | OUTPUT - The priority is returned in the DBGEXTSW_PRIORITY field.
0654 0 |
0655 0 | DBGEXTSK_GET_PRIORITY = 12.
0656 0 |
0657 0 |
0658 0 | SET_PRIORITY is used to set the priority of a specified task.
0659 0 |
0660 0 | INPUTS - The task value is placed in the DBGEXTSL_TASK_VALUE field.
0661 0 |
0662 0 | If the TASK_VALUE field is zero and the ALL flag
0663 0 | is set, then the function is done for all tasks.
0664 0 |

0665 0 | The desired priority is placed in the DBGEXTSW_PRIORITY field.
0666 0
0667 0 | OUTPUT - none.
0668 0
0669 0 | DBGEXTSK_SET_PRIORITY = 13.
0670 0
0671 0
0672 0 | RESTORE_PRIORITY is used to restore the priority of a task back
0673 0 to its normal value (as it would be without DEBUG intervention).
0674 0
0675 0 | INPUTS - The task value is placed in the DBGEXTSL_TASK_VALUE field.
0676 0
0677 0 | If the TASK_VALUE field is zero and the ALL flag
0678 0 is set, then the function is done for all tasks.
0679 0
0680 0 | OUTPUT - none.
0681 0
0682 0 | DBGEXTSK_RESTORE_PRIORITY = 14.
0683 0
0684 0
0685 0 | GET_REGISTERS is used to obtain the register set of a task.
0686 0
0687 0 | INPUT - The task value is placed in the DBGEXTSL_TASK_VALUE field.
0688 0
0689 0 | OUTPUTS - The register values are returned in the DBGEXTSL_R0
0690 0 through DBGEXTSL_PSL fields.
0691 0
0692 0 | NOTE: Only DEBUG knows the register set of the active task
0693 0 hence, this call is invalid for the active task.
0694 0 A return status of STSSK_SEVERE is returned.
0695 0
0696 0 | DBGEXTSK_GET_REGISTERS = 15.
0697 0
0698 0
0699 0 | SET_REGISTERS is used to change the register values of a task.
0700 0 This may be needed, for example, in SET TASK T;DEPOSIT R5 = 0;60
0701 0
0702 0 | INPUTS - The task value is placed in the DBGEXTSL_TASK_VALUE field.
0703 0
0704 0 | The register values are placed in the DBGEXTSL_R0
0705 0 through DBGEXTSL_PSL fields.
0706 0
0707 0 | OUTPUT - none.
0708 0
0709 0 | NOTE: Only DEBUG knows the register set of the active task
0710 0 hence, this call is invalid for the active task.
0711 0 A return status of STSSK_SEVERE is returned.
0712 0
0713 0 | DBGEXTSK_SET_REGISTERS = 16.
0714 0
0715 0
0716 0 | ENABLE EVENT is used during processing of a "SET BREAK/EVENT=" or
0717 0 "SET TRACE/EVENT=" command to enable reporting of a given kind of event.
0718 0
0719 0 | INPUTS - The DBGEXTSL EVENT_ID field contains a code identifying
0720 0 the event being enabled. The possible values of this
0721 0 code are defined below.

0722 0
0723 0
0724 0
0725 0
0726 0
0727 0
0728 0
0729 0
0730 0
0731 0
0732 0
0733 0
0734 0
0735 0
0736 0
0737 0
0738 0
0739 0
0740 0
0741 0
0742 0
0743 0
0744 0
0745 0
0746 0
0747 0
0748 0
0749 0
0750 0
0751 0
0752 0
0753 0
0754 0
0755 0
0756 0
0757 0
0758 0
0759 0

The DBGEXTSL TASK_VALUE field contains a task value further qualifying the event being enabled. This may be zero if the "ALL" flag is lit.

For example, if we are enabling "task termination" and we supply a task value, then we only want to break on termination of that task. If we enable "task termination" events and set the ALL flag, we want to be notified of any task termination.

OUTPUT - none

DBGEXTSK_ENABLE_EVENT = 17.

DISABLE_EVENT is used during processing of a "CANCEL BREAK/EVENT=" or "CANCEL TRACE/EVENT=" command to disable reporting of a given kind of event.

INPUTS - The DBGEXTSL EVENT_ID field contains a code identifying the event being disabled. The possible values of this code are defined below.

The DBGEXTSL TASK_VALUE field contains a task value further qualifying the event being disabled. This may be zero if the "ALL" flag is lit.

OUTPUT - none

DBGEXTSK_DISABLE_EVENT = 18.

A maximum task code is defined for CASE statement bounds.

DBGEXTSK_MAX_FUNCT = 18;

0760 0
0761 0
0762 0
0763 0
0764 0
0765 0
0766 0
0767 0
0768 0
0769 0
0770 0
0771 0
0772 0
0773 0
0774 0
0775 0
0776 0
0777 0
0778 0
0779 0
0780 0
0781 0
0782 0
0783 0
0784 0
0785 0
0786 0
0787 0
0788 0
0789 0
0790 0
0791 0
0792 0
0793 0
0794 0
0795 0
0796 0
0797 0
0798 0
0799 0
0800 0
0801 0
0802 0
0803 0
0804 0
0805 0
0806 0
0807 0
0808 0
0809 0
0810 0
0811 0
0812 0
0813 0
0814 0
0815 0
0816 0

COMPLETION STATUS

The run time system has two means of providing a completion status -- the return value of the function and the contents of `DBGEXTSL_STATUS`.

Function Return Value --

The run time system should, as its first action, attempt to read and verify the field `DBGEXTSV FACILITY_ID` in `DBGEXT$CONTROL_BLOCK`. Optionally, it may also PROBE the control block for read/writability. If the `FACILITY_ID` is correct, the run time system should eventually return:

`STSSK_SUCCESS` - service successfully completed

Otherwise, the run time system should immediately return:

`STSSK_SEVERE` - service failed

This helps to insure that an incorrect External Control Block will be detected before it is written to.

Contents of `DBGEXTSL_STATUS` --

All other status and error conditions will be placed in the `STATUS` field of the control block. The possible values of the `STATUS` field are a composite of severity level and message number. Only two severity values are used. They are given by `STSSV_SEVERITY`:

`STSSK_SUCCESS` - service successfully completed

In this case the message number (`STSSV_MSG_NO`) is zero.

`STSSK_ERROR` - service failed

In this case the message number (`STSSV_MSG_NO`) is one of the following:

LITERAL

`DBGEXTSK_FUNCTION_NOT_IMP` = 0.

The function requested is not implemented by the facility.

`DBGEXTSK_TASK_NOT_EXIST` = 1.

Task number cannot be translated to a task value because the task does not exist. Or task value does not point to a currently existing task (this cannot always be detected).

`DBGEXTSK_TASK_IS_ACTIVE` = 2.

Returned on a `SET_REGISTER` or `GET_REGISTER` function for the active task. The run time system cannot access the registers of the active task.

`DBGEXTSK_TASK_IS_NULL` = 3:

L 16
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_ \$255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 20
(11)

: 0817 0 ! Returned on a SET_ACTIVE function for the null task.
: 0818 0

0819 0
0820 0
0821 0
0822 0
0823 0
0824 0
0825 0
0826 0
0827 0
0828 0
0829 0
0830 0
0831 0
0832 0
0833 0
0834 0
0835 0
0836 0
0837 0
0838 0
0839 0
0840 0
0841 0
0842 0
0843 0
0844 0
0845 0
0846 0
0847 0
0848 0
0849 0
0850 0
0851 0
0852 0
0853 0
0854 0
0855 0
0856 0
0857 0
0858 0
0859 0
0860 0
0861 0
0862 0
0863 0
0864 0
0865 0
0866 0
0867 0
0868 0
0869 0
0870 0
0871 0

PRINT ROUTINE INTERFACE

The following defines how to use the DEBUG print routine whose address is given in the DBGEXT\$L_PRINT_ROUTINE field.

BIND

```
DBG$PRINT_ROUTINE = .control_block [ DBG$L_PRINT_ROUTINE ];  
DBG$PRINT_ROUTINE ( NEW_LINE,  
                     STRING_TO_PRINT,  
                     FAO_ARG_1,  
                     FAO_ARG_2...  
                     FAO_ARG_n ) : NOVALUE
```

NEW_LINE - this can have one of two values:

- 0 - Place the given string in the output buffer.
- 1 - If the given string is non-zero, first place it in the buffer. In all cases, output the buffer to the screen.

STRING_TO_PRINT

- this is a pointer to a counted ascii string
E.g., UPLIT (\$ASCIC 'Output this text')
This may be zero if the ACTION_CODE is 'NEWLINE'.

There may be FAO arguments following the string.
The string thus may contain embedded FAO commands
such as '!AC', '!SL', and so on.

%((FIXUP - THIS EXTENSION IS NOT GOOD!! %))%

In addition, there will be a DEBUG-specific extension
to FAO which can be used for symbolizing addresses.
There will be a new command '!SA' for "symbolize address".
This indicates that the corresponding FAO argument
is an address. Its symbolization is to be embedded into
the string.

FAO_ARG1 through FAO_ARGn - optional parameters for FAO arguments.

Example: suppose FOO\L is located at address 200. Then:

```
DBG$PRINT_ROUTINE (DBGEXT$K_NEWLINE,  
                    UPLIT ($ASCIC 'Task switch at location !SA'),  
                    200);
```

This would output:

"Task switch at location FOO\L"

!-

8 1
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 22
(13)

0872 0 |
0873 0 |
0874 0 | The following define the possible values of the DBGEXTSL_EVENT_ID field.
0875 0 | These are the predefined events that we can break or trace on.
0876 0 |
0877 0 | LITERAL
0878 0 | DBGEXTSK_MIN_EVENT_CODE = 0.
0879 0 |
0880 0 | DBGEXTSK_INVOKE_DEBUG = 0. ! Unconditional DEBUG invocation
0881 0 |
0882 0 |
0883 0 | DBGEXTSK_TASK_ACTIVATION = 1. ! First transition of a task to RUNNING
0884 0 | DBGEXTSK_TASK_SUSPENSION = 2. ! Transition from RUNNING to SUSPENDED
0885 0 | DBGEXTSK_TASK_SWITCH_FROM = 3. ! Transition from RUNNING to some state
0886 0 | DBGEXTSK_TASK_SWITCH_TO = 4. ! Transition from some state to RUNNING
0887 0 | DBGEXTSK_TASK_TERMINATION = 5. ! Any kind of termination
0888 0 |
0889 0 | Ada specific tasking codes:
0890 0 |
0891 0 | DBGEXTSK_TASK_ABORT_TERM = 6. ! Termination by abort
0892 0 | DBGEXTSK_TASK_EXCEP_TERM = 7. ! Termination by unhandled exception
0893 0 | DBGEXTSK_TASK_EXCEP_RENDER = 8. ! Exception propagating out of rendezvous
0894 0 | DBGEXTSK_TASK_ENTRY_CALL = 9. ! Executing an entry call
0895 0 | DBGEXTSK_TASK_ACCEPT = 10. ! Executing an accept
0896 0 | DBGEXTSK_TASK_SELECT = 11. ! Executing a select
0897 0 |
0898 0 | DBGEXTSK_MAX_EVENT_CODE = 11;
0899 0 |

{
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742

_S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 23
(14)

0900 0

EVENT CONTROL BLOCK

0901 0

The Event Control Block is the data structure that the ADA (or other) facility passes to DEBUG when it signals that a given event has occurred.

0902 0

For example, if you do a SET BREAK/ADAEVENT=TASK_SWITCH_TO, then when a task switch occurs, the ADA run-time system will signal the special signal DBGS_EVENT. A pointer to an "Event Control Block" is passed as the "FAO argument" of DBGS_EVENT. (E.g., LIB\$SIGNAL (DBGS_EVENT, 1, .EVENT CONTROL BLOCK). (Note that this condition cannot properly be an SSS condition because they are not allowed to have FAO arguments other than PC and PSL (except for the hardware conditions). Hence, the facility DBG was chosen. This condition is a DEBUG-defined condition that anyone can signal. The FAO count of 1 is required so that the message conforms to a legal format for a message vector.) Through proper use of the SEVERITY field and the NODMESSAGE bit in the condition, the signaller can be assured that events will be "reflected" by Traceback should DEBUG not be mapped into the image (for some reason). So there really are no restrictions on when this condition can be signalled.

0903 0

The control block contains a code indicating the facility that has originated the event and another code to indicate what event has occurred. It also contains message text to be output announcing the event.

0904 0

0905 0

0906 0

0907 0

0908 0

0909 0

0910 0

0911 0

0912 0

0913 0

0914 0

0915 0

0916 0

0917 0

0918 0

0919 0

0920 0

0921 0

0922 0

0923 0

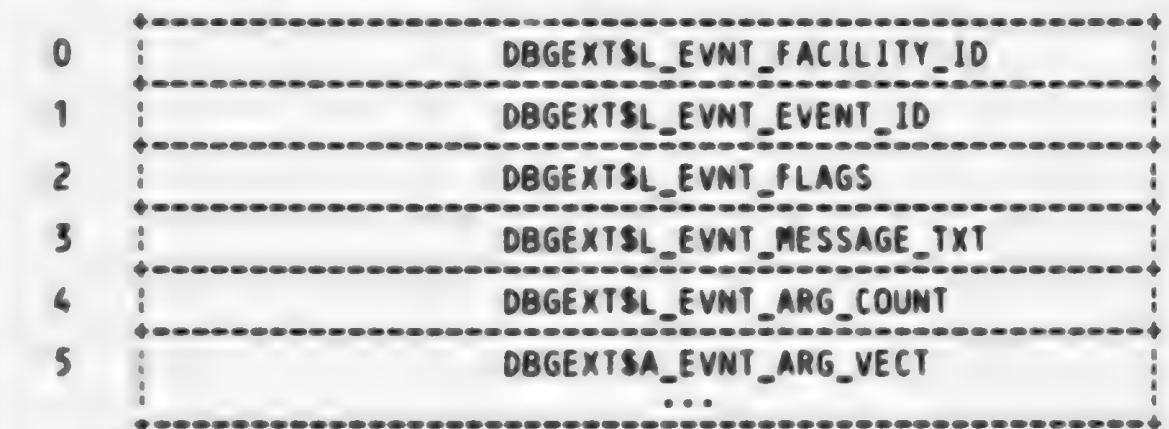
0924 0

0925 0

0926 0

0927 0

The following illustrates the Event Control Block:



FIELD DBGEXTSEVNT_FIELDS =

SET

DBGEXTSL_EVNT_FACILITY_ID	= [0, 0, 32, 0],
DBGEXTSL_EVNT_EVENT_ID	= [1, 0, 32, 0],
DBGEXTSL_EVNT_FLAGS	= [2, 0, 32, 0],
DBGEXTSV_EVNT_MORE_TEXT	= [2, 0, 1, 0], ! Flag bit 0
DBGEXTSV_EVNT_REENTRY	= [2, 1, 1, 0], ! Flag bit 1
DBGEXTSL_EVNT_MESSAGE_TXT	= [3, 0, 32, 0],
DBGEXTSL_EVNT_ARG_COUNT	= [4, 0, 32, 0],
DBGEXTSA_EVNT_ARG_VECT	= [5, 0, 0, 0]

TES;

D 1
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 24
(14)

0957 0 LITERAL
0958 0 DBGEXTSK_EVNT_BASE_SIZE = 5;
0959 0
0960 0 MACRO
M 0961 0 DBGEXTSEVENT CONTROL_BLOCK(NUM_ARGS) =
M 0962 0 BLOCK [DBGEXTSK BASE_SIZE + NUM_ARGS ,LONG]
M 0963 0 FIELD (DBGEXTSEVNT_FIELDS);
0964 0
0965 0
0966 0 Explanation of fields:
0967 0
0968 0 FACILITY_ID field: The code for the facility signaling the
0969 0 event. If the CUST_DEF bit is set the
0970 0 event is a "user event". Otherwise, the
0971 0 only supported codes are ADA, PPA, and
0972 0 scan.
0973 0
0974 0 EVENT_ID field: This field contains the event code.
0975 0 Event codes are numbered from 1 within
0976 0 each facility. Event code 0 is
0977 0 reserved in all facilities. It represents
0978 0 the unconditional event, that is,
0979 0 unconditional DEBUG entry. If the
0980 0 EVENT_ID field is zero, the REENTRY bit
0981 0 is checked.
0982 0
0983 0 MESSAGE_TXT field: This is a pointer to a counted ascii string.
0984 0 The string represents a message to be printed
0985 0 when the event occurs and is formatted as an
0986 0 "fao control string". The string may take FAO
0987 0 arguments. The string may also contain the
0988 0 DEBUG extension to FAO, '!SA', in order to
0989 0 symbolize an address. This extension is
0990 0 described above. NOTE: if this field is 0,
0991 0 it indicates that there is no message.
0992 0
0993 0 ARG_COUNT field: Count of the number of FAO arguments that go
0994 0 with the text.
0995 0
0996 0 ARG_VECT field: A vector of FAO arguments.
0997 0
0998 0 MORE_TEXT flag: If this flag is TRUE, it indicates that DEBUG
0999 0 is to return control at the point of the signal
1000 0 after displaying the message. This is to be used
1001 0 for output of multi-line messages. (I.e., the
1002 0 run-time system should then resignal the event with
1003 0 the next line of message text in the MESSAGE_TXT
1004 0 field).
1005 0
1006 0 REENTRY flag: If this flag is TRUE, then this event is a
1007 0 DEBUG-reentry event that has occurred after a
1008 0 PSEUDO GO. DEBUG is thereby instructed
1009 0 to restore certain components of its state
1010 0 from the values they had at DEBUG's last
1011 0 incarnation (e.g. AST enablement).
1012 0 For this flag to be checked by DEBUG, the
1013 0 EVENT_ID field MUST BE ZERO, thus indicating

E 1
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
\$255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 25
(14)

: 1014 0 !
: 1015 0 unconditional entry to DEBUG.

1016 0

1017 0

1018 0

1019 0

1020 0

1021 0

1022 0

1023 0

1024 0

1025 0

1026 0

1027 0

1028 0

1029 0

1030 0

1031 0

1032 0

1033 0

1034 0

1035 0

1036 0

1037 0

1038 0

1039 0

1040 0

1041 0

1042 0

1043 0

1044 0

1045 0

1046 0

1047 0

1048 0

1049 0

1050 0

1051 0

1052 0

1053 0

1054 0

1055 0

1056 0

1057 0

1058 0

1059 0

1060 0

1061 0

1062 0

1063 0

1064 0

1065 0

1066 0

1067 0

1068 0

1069 0

1070 0

1071 0

REGISTERING EVENTS WITH DEBUG

DEBUG's event handling feature is available to user programs as well as Digital software. DEBUG maintains an event table for each facility that chooses to register its events with DEBUG.

Registering an event with DEBUG is very simple. The facility need only signal the following signal after DEBUG has been invoked in an image:

```
LIB$SIGNAL(DBGS_REGISTER_EVENTS,  
           first_event_condition,  
           second_event_condition,  
           etc.)
```

A list of event conditions is chained below a master condition of DBGS_REGISTER_EVENTS. This signal may be raised as many times as desired to add more events to DEBUG's event table. Since DEBUG derives the facility number from the event condition, events for different facilities may be registered with the same signal.

The event conditions appearing in the message vector must be defined in the facilities message file. The string defined in the message file is the string that DEBUG will use to name the event.

For example, suppose we wish to add an event of PLI\$_TASK_SWITCH. The following would do it:

1. Add to PLI's message file:

```
PLI$_FACILITY = xxx  
TASK_SWITCH    "TASK_SWITCH"
```

2. Register the event with DEBUG

```
LIB$SIGNAL(DBGS_REGISTER_EVENTS, PLI$_TASK_SWITCH)
```

After the registration, any user can then type

```
SET BREAK/EVENT=PLI$_TASK_SWITCH
```

A command SET EVENT/FACILITY='PLI\$' can be used so the facility prefix can be omitted, e.g. SET BREAK/EVENT=TASK_SWITCH. This will then not be confused with an Ada task switch. SET EVENT/NOFACILITY will eliminate the automatic prefixing of event names.

To simplify the registration of events by facilities, any facility should provide an entry point that users can call from the DEBUGGER to load the events of that facility. To load PLI's events, then, a user would merely type

```
DBG> CALL PLI$LOAD_EVENTS
```

** Obviously, Ada's events should be registered with this same general mechanism

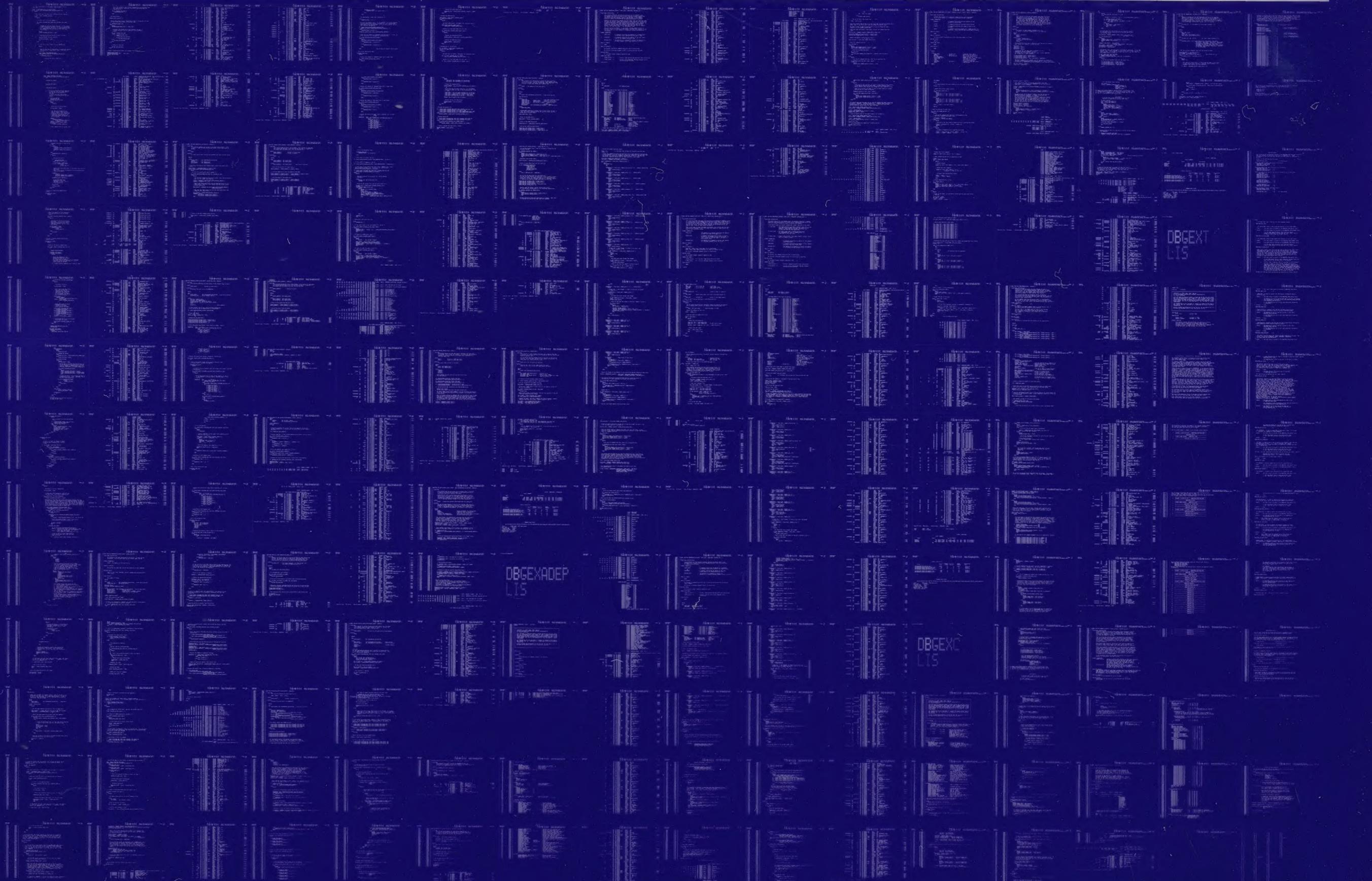
: COMMAND QUALIFIERS

: BLISS/LIBRARY=LIB\$:DBGEXT.L32/LIST=LIS\$:DBGEXT.LIS SRC\$:DBGEXT.REQ

: Run Time: 00:06.5
: Elapsed Time: 00:09.6
: Lines/CPU Min: 9962
: Lexemes/CPU-Min: 15106
: Memory Used: 38 pages
: Library Precompilation Complete

0083 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



0084 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

DBGIFTHEN
LIS

DBGLANVEC
LIS

DBGGEN
LIS

DBGLANGOP
LIS

DBGLEVEL
LIS